A APPENDIX

A.1 Notations

Table 6: Frequently used notations in this paper.

Notations	Descriptions
$\mathcal{G}^{(t)}$	Directed graph at time t
$n^{(t)}, m^{(t)}$	Numbers of nodes and edges at time t
F	Dimension of the node attribute
$\boldsymbol{X}^{(t)}$	The node attribute matrix and $\boldsymbol{X}^{(t)} \in \mathbb{R}^{n^{(t)} \times F}$
$\boldsymbol{x}_i^{(t)}$	The attribute vector of the i-th dimension
$\mathcal{N}_{out}^{(t)}(v), \mathcal{N}_{in}^{(t)}(v)$	The out-neighbors and in-neighbors of v
$e_t = \{u_t, v_t\}, \Gamma$	The update event and the event set
$\boldsymbol{r}^{(t)}$	Residue vector at time t and $\boldsymbol{r}^{(t)} \in \mathbb{R}^{n^{(t)} \times 1}$
$\boldsymbol{h}^{(t)}$	Reserve vector at time t and $\boldsymbol{h}^{(t)} \in \mathbb{R}^{n^{(t)} \times 1}$
$\boldsymbol{H}^{(t)}$	The node embedding matrix at time t
$\boldsymbol{M}^{(t)}$	The state matrix at time t
$\bar{\mathcal{A}}, \bar{\mathcal{B}}, C$	The network parameters in SSM
α	Teleport probability of random walks
λ	Threshold in to control the embedding distance

A.2 Proof of Lemma 1

Proof. According to the definition of 1-Norm of the matrix, we have:

$$\begin{aligned}||\boldsymbol{H}^{(t+1)} - \boldsymbol{H}^{(t)}||_1 &= \max_{f \in F} \sum_u |\boldsymbol{h}^{(t+1)}(u)) - \boldsymbol{h}^{(t)}(u))| \\ &\leq \max_{f \in F} \sum_u |\boldsymbol{z}^{(t+1)}(u) - \boldsymbol{z}^{(t)}(u)) + 2\epsilon|\end{aligned}$$

We can express $\boldsymbol{Z}^{(t)}$ as the result of global smoothness [82, 83] which satisfies:

$$\boldsymbol{Z}^{(t)} = \left(\boldsymbol{I} + c\boldsymbol{L}^{(t)}\right)^{-1} \boldsymbol{X}^{(t)}, \tag{6}$$

where $c = \frac{1}{\alpha} - 1$ and $\boldsymbol{L}^{(t)} = \boldsymbol{I} - \boldsymbol{P}^{(t)}$ is the Laplacian matrix and denotes the topological status of graph $\mathcal{G}^{(t)}$. Hence we have:

$$\begin{aligned}&||\boldsymbol{H}^{(t+1)} - \boldsymbol{H}^{(t)}||_1 \\ &\leq \max_{f \in F} \left\| \left(\left(\boldsymbol{I} + c\boldsymbol{L}^{(t+1)}\right)^{-1} - \left(\boldsymbol{I} + c\boldsymbol{L}^{(t)}\right)^{-1} \right) \boldsymbol{x}^{(t)} \right\|_1 + 2n\epsilon \\ &\leq c \max_{f \in F} \left\| \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+1)}\right)^{-1} \left(\boldsymbol{L}^{(t+1)} - \boldsymbol{L}^{(t)}\right) \left(\boldsymbol{I} + c\boldsymbol{L}^{(t)}\right)^{-1} \boldsymbol{x}^{(t)} \right\|_1 \\ &+ 2n\epsilon \leq c \max_{f \in F} \left\| \left(\boldsymbol{P}^{(t+1)} - \boldsymbol{P}^{(t)}\right) \boldsymbol{x}^{(t)} \right\|_1 + 2n\epsilon \\ &\leq \frac{1-\alpha}{\alpha} \|\Delta \boldsymbol{P} \cdot \boldsymbol{x}_{max}\|_1 + 2n\epsilon,\end{aligned}$$

where $\boldsymbol{x}_{max}$ is the row-wise maximum absolute value vector and i-th entry of $\boldsymbol{x}_{max}$ is defined as: $\{\boldsymbol{x}_{max}\}_i = \max_{1 \leq j \leq n} |\boldsymbol{X}_{ij}|$. Proof finished.

□

A.3 Proof of Proposition 1

Proof. We assume there come p edge updates between t and $t+1$. Since we can not access the concrete timestamps of these p edges, we can assume k-th each edge arrives at the system at a hidden timestamp $t + \tau_k$ $(1 \leq k \leq p)$ without loss of generality. Specifically, we have $t + \tau_p = t + 1$. According to [56], the complete evolving process from t to $t+1$ can be formulated as:

$$\begin{aligned}\boldsymbol{M}^{(t+1)} &= \bar{\mathcal{A}}\boldsymbol{M}^{(t)} + \bar{\mathcal{B}} \sum_{k=1}^{p} \sum_{l=0}^{\infty} \alpha \left((1-\alpha)^l \boldsymbol{P}^{(t+\tau_k)}\right)^l \boldsymbol{X}\Lambda_k \\ &= \bar{\mathcal{A}}\boldsymbol{M}^{(t)} + \bar{\mathcal{B}} \sum_{k=1}^{p} \alpha \left(\boldsymbol{I} - (1-\alpha)\boldsymbol{P}^{(t+\tau_k)}\right)^{-1} \boldsymbol{X}\Lambda_k.\end{aligned}$$

When setting $\alpha = \frac{1}{c+1}$, we have:

$$\boldsymbol{M}^{(t+1)} = \bar{\mathcal{A}}\boldsymbol{M}^{(t)} + \bar{\mathcal{B}} \sum_{k=1}^{p} \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+\tau_k)}\right)^{-1} \boldsymbol{X}\Lambda_k,$$

where $\boldsymbol{L}^{(t)}$ is the Laplacian matrix at time t. Given the exact PPR embedding $\left(\boldsymbol{I} + c\boldsymbol{L}^{(t+1)}\right)^{-1} \boldsymbol{X}$ at time $t+1$, we have:

$$\begin{aligned}&\left\| \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+1)}\right)^{-1} \boldsymbol{X} - \sum_{k=1}^{p} \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+\tau_k)}\right)^{-1} \boldsymbol{X}\Lambda_k \right\| \\ &= \left\| \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+1)}\right)^{-1} \boldsymbol{X} \sum_{k=1}^{p} \Lambda_k - \sum_{k=1}^{p} \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+\tau_k)}\right)^{-1} \boldsymbol{X}\Lambda_k \right\| \\ &= \left\| \sum_{k=1}^{p} \left(\left(\boldsymbol{I} + c\boldsymbol{L}^{(t+1)}\right)^{-1} - \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+\tau_k)}\right)^{-1} \right) \boldsymbol{X}\Lambda_k \right\| \\ &\leq \left\| \sum_{k=1}^{p} \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+1)}\right)^{-1} \left(c\boldsymbol{L}^{(t+1)} - c\boldsymbol{L}^{(t+\tau_k)}\right) \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+\tau_k)}\right)^{-1} \boldsymbol{X}\Lambda_k \right\| \\ &\leq c \left\| \sum_{k=1}^{p} \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+1)}\right)^{-1} \left(\boldsymbol{L}^{(t+1)} - \boldsymbol{L}^{(t+\tau_k)}\right) \left(\boldsymbol{I} + c\boldsymbol{L}^{(t+\tau_k)}\right)^{-1} \boldsymbol{X} \right\| \left\| \sum_{k=1}^{p} \Lambda_k \right\| \\ &\leq c \sum_{k=1}^{p} \frac{\left\| \left(\boldsymbol{L}^{(t+1)} - \boldsymbol{L}^{(t+\tau_k)}\right) \boldsymbol{X} \right\|_2}{\left(1 + c\lambda_1(\boldsymbol{L}^{(t+1)})\right)\left(1 + c\lambda_1(\boldsymbol{L}^{(t+\tau_k)})\right)} \\ &\leq c \sum_{k=1}^{p} \left\| \left(\boldsymbol{L}^{(t+1)} - \boldsymbol{L}^{(t+\tau_k)}\right) \boldsymbol{X} \right\| \leq \lambda\end{aligned}$$

□

A.4 Proof of Proposition 2

Proof. Given $\boldsymbol{M}^{(0)} = \bar{\mathcal{A}}^{(0)} \cdot \boldsymbol{0} + \bar{\mathcal{B}}^{(0)} \cdot \boldsymbol{H}^{(0)} = \bar{\mathcal{B}}^{(0)} \boldsymbol{H}^{(0)}$ where $\bar{\mathcal{A}}^{(t)}$ and $\bar{\mathcal{B}}^{(t)}$ denote the parameter metrices at time t. Following the state update principle [4] of Equ. 3, the state matrix at t can be formulated as:

$$\begin{aligned}\boldsymbol{M}^{(t)} =& \bar{\mathcal{A}}^{(t)} \bar{\mathcal{A}}^{(t-1)} \ldots \bar{\mathcal{A}}^{(1)} \bar{\mathcal{B}}^{(0)} \boldsymbol{H}^{(0)} + \bar{\mathcal{A}}^{(t)} \bar{\mathcal{A}}^{(t-1)} \ldots \bar{\mathcal{A}}^{(2)} \bar{\mathcal{B}}^{(1)} \boldsymbol{H}^{(1)} + \\ &\ldots + \bar{\mathcal{A}}^{(t)} \bar{\mathcal{A}}^{(t-1)} \bar{\mathcal{B}}^{(t-2)} \boldsymbol{H}^{(t-2)} + \bar{\mathcal{A}}^{(t)} \bar{\mathcal{B}}^{(t-1)} \boldsymbol{H}^{(t-1)} + \bar{\mathcal{B}}^{(t)} \boldsymbol{H}^{(t)} \\ &= \sum_{s=0}^{t} \prod_{i=s+1}^{t} \bar{\mathcal{A}}^{(i)} \bar{\mathcal{B}}^{(s)} \boldsymbol{H}^{(s)},\end{aligned} \tag{7}$$

where we define $\prod_{i=t+1}^{t} \bar{\mathcal{A}}^{(i)} = \boldsymbol{I}$. When we vectorize the prediction result over time $[0, t]$ and restructure the mathematical

[4] Without the loss of generalization, here we simplify the notation and assume the sample time corresponds with the prediction time for a clear presentation.

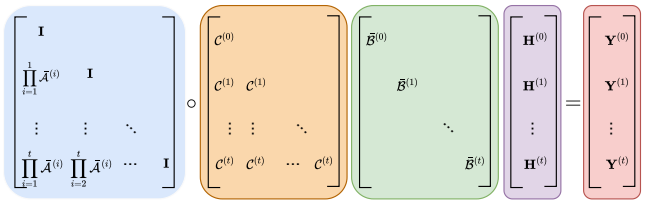


Figure 4: The equivalent kernel attention in CODEN.

expression, we can derive the formula representing our prediction across all time steps:

$$Y^{(0:t)} = \left(\boldsymbol{L} \circ \boldsymbol{Q}\boldsymbol{K}^\top\right) \cdot \boldsymbol{V}, \tag{8}$$

where (i) $\boldsymbol{L}$ is a lower-triangular matrix and $\boldsymbol{L}_{uv} = \prod_{i=v+1}^{u} \bar{\mathcal{A}}^{(i)}$; (ii) $\boldsymbol{Q}$ is a lower-triangular matrix and $\boldsymbol{Q}_{uv} = C^{(u)}$, where C is the parameters of MLP$(\cdot)$; (iii) $\boldsymbol{K}$ is a diagonal matrix and $\boldsymbol{K}_{uu} = \bar{\mathcal{B}}^{(u)}$; (iv) $\boldsymbol{V}$ is the vectorized node embedding generated at all time steps and $\boldsymbol{V}(u) = \boldsymbol{H}^{(u)}$.

As illustrated in Fig. 4, we observe that the fundamental methodology of CODEN for processing temporal information is fully aligned with the definition of the kernel attention mechanism [72], which is developed to mitigate the "over attention" phenomenon [76] in computer vision [74] and natural language processing (NLP) areas [75]. Upon revisiting the motivation behind kernel attention in NLP, where it is crucial to selectively maintain and degrade specific tokens [75], it becomes apparent that the trainable matrices $\bar{\mathcal{A}}^{(i)}$ $(0 \le i \le t)$ play a pivotal role. □

A.5 Proof of Lemma 2

PROOF. Based on the definition of the Dirichlet Energy and simplifying $\boldsymbol{I} - \boldsymbol{A}^{(t)\top}\boldsymbol{D}^{(t)^{-1}}$ as Δ, we have:

$$\mathbb{DE}(\boldsymbol{M}_A^{(t)}) = \mathbb{DE}\left(\text{softmax}\left(\frac{\left(\boldsymbol{W}_q\boldsymbol{H}^{(t)}\right)\cdot\left(\boldsymbol{W}_k\boldsymbol{H}^{(s)}\right)^\top}{\sqrt{F'}}\right)_{0:t} \cdot \left(\boldsymbol{W}_v\boldsymbol{H}^{(s)}\right)_{0:t}\right).$$

To simplify the demonstration, we denote

$$\boldsymbol{S}_t = \text{softmax}\left(\left(\boldsymbol{W}_q\boldsymbol{H}^{(t)}\right)\cdot\left(\boldsymbol{W}_k\boldsymbol{H}^{(s)}\right)^\top\right)_{0:t}$$

. Since $\boldsymbol{S}_t$ is a Row-Stochastic matrix where the sum of each row equals to 1, we can obtain the maximum eigenvalue $\sigma_{max}(\boldsymbol{S}_t)$ of $\boldsymbol{S}_t$ as $\sigma_{max}(\boldsymbol{S}_t) \le 1$. Then we have:

$$\begin{aligned}
\mathbb{DE}(\boldsymbol{M}_A^{(t)}) &= tr\left(\boldsymbol{S}_t\left(\boldsymbol{H}^{(s)}\right)_{0:t}\Delta\boldsymbol{S}_t^\top\left(\boldsymbol{H}^{(s)}\right)_{0:t}^\top\right)/F \\
&= tr\left(\left(\boldsymbol{H}^{(s)}\right)_{0:t}\Delta\left(\boldsymbol{H}^{(s)}\right)_{0:t}^\top\boldsymbol{S}_t\boldsymbol{S}_t^\top\right)/F \\
&\le tr\left(\left(\boldsymbol{H}^{(s)}\right)_{0:t}\Delta\left(\boldsymbol{H}^{(s)}\right)_{0:t}^\top\right)\sigma_{max}(\boldsymbol{S}_t\boldsymbol{S}_t^\top)/F \\
&\le tr\left(\left(\boldsymbol{H}^{(s)}\right)_{0:t}\Delta\left(\boldsymbol{H}^{(s)}\right)_{0:t}^\top\right)/F \\
&= \sum_{s=0}^{t} tr\left(\left(\boldsymbol{H}^{(s)}\right)\left(\boldsymbol{H}^{(s)}\right)^\top\Delta\right)/F
\end{aligned}$$

Following the similar derivation, we can obtain the Dirichlet Energy $\mathbb{DE}(\boldsymbol{M}_C^{(t)})$ as:

$$\begin{aligned}
\mathbb{DE}(\boldsymbol{M}_C^{(t)}) &\ge \sum_{s=0}^{t} tr\left(\prod_{i=s+1}^{t}\bar{\mathcal{A}}^{(i)}\bar{\mathcal{B}}^{(s)}\left(\boldsymbol{H}^{(s)}\right)\left(\boldsymbol{H}^{(s)}\right)^\top\Delta\right) \\
&\ge \sum_{s=0}^{t}\sigma_{min}\left(\prod_{i=s+1}^{t}\bar{\mathcal{A}}^{(i)}\bar{\mathcal{B}}^{(s)}\right)tr\left(\left(\boldsymbol{H}^{(s)}\right)\left(\boldsymbol{H}^{(s)}\right)^\top\Delta\right) \\
&\ge \mathbb{DE}(\boldsymbol{M}_A^{(t)}),
\end{aligned}$$

which holds when $F\cdot\sigma_{min}^2\left(\prod_{i=s+1}^{t}\bar{\mathcal{A}}^{(i)}\bar{\mathcal{B}}^{(s)}\right) \ge 1$ for $0 \le s \le t$. □

A.6 Proof of Lemma 5

PROOF. According to [32], the estimated vector $\hat{h}^{(t-1)}$ at time $t-1$ and the exact PPR vector $\sum_{l=0}^{\infty}\alpha\left((1-\alpha)^l\boldsymbol{P}^{(t-1)}\right)^l\cdot\boldsymbol{x}_i$ have the following relationship:

$$\begin{aligned}
\sum_{l=0}^{\infty}\alpha\left((1-\alpha)^l\boldsymbol{P}^{(t-1)}\right)^l\boldsymbol{x}_i &= \hat{h}^{(t-1)} + \sum_{l=0}^{\infty}\alpha\left((1-\alpha)^l\boldsymbol{P}^{(t-1)}\right)^l\boldsymbol{r}_i \\
&= \hat{h}^{(t-1)} + \alpha\left(I-(1-\alpha)\boldsymbol{P}^{(t-1)}\right)^{-1}\boldsymbol{r}_i
\end{aligned}$$

For the purpose of clear demonstration, we express $(1-\alpha)P^{(t)}$ as $P^{(t)}$ for the following proof. Then we formulate the output $\hat{h}^{(t)}$ of Alg. 3 as:

$$\begin{aligned}
\hat{h}^{(t)} &= \sum_{j=0}^{\infty}(P^{(t)})^j(P^{(t)}-P^{(t-1)})\ (\alpha\sum_{i=0}^{\infty}(P^{(t-1)})^i\boldsymbol{x}_i - \alpha\sum_{i=0}^{\infty}(P^{(t-1)})^i \\
&\quad \boldsymbol{r}_i^{(t-1)}) + \hat{h}^{(t-1)} \\
&= \sum_{j=0}^{\infty}(P^{(t)})^j(P^{(t)}-P^{(t-1)})\alpha\sum_{i=0}^{\infty}(P^{(t-1)})^i\boldsymbol{x}_i - \sum_{j=0}^{\infty}(P^{(t)})^j(P^{(t)}- \\
&\quad P^{(t-1)})\alpha\sum_{i=0}^{\infty}(P^{(t-1)})^i\boldsymbol{r}_i^{(t-1)} + \hat{h}^{(t-1)}
\end{aligned} \tag{9}$$

The first term of the last line in Equ. 9 can be simplified as:

$$\begin{aligned}
&\sum_{j=0}^{\infty}(P^{(t)})^j(P^{(t)}-P^{(t-1)})\alpha\sum_{i=0}^{\infty}(P^{(t-1)})^i\boldsymbol{x}_i \\
&= \sum_{j=1}^{\infty}(P^{(t)})^j\alpha\sum_{i=0}^{\infty}(P^{(t-1)})^i\boldsymbol{x}_i - \sum_{j=0}^{\infty}(P^{(t)})^j\alpha\sum_{i=j}^{\infty}(P^{(t-1)})^i\boldsymbol{x}_i \\
&= \sum_{j=1}^{\infty}(P^{(t)})^j\alpha\left(\sum_{i=1}^{\infty}(P^{(t-1)})^i + \boldsymbol{I}\right)\boldsymbol{x}_i - \left(\sum_{j=1}^{\infty}(P^{(t)})^j + \boldsymbol{I}\right)\alpha\sum_{i=1}^{\infty}(P^{(t-1)})^i\boldsymbol{x}_i \\
&= \alpha\sum_{j=1}^{\infty}(P^{(t)})^i\boldsymbol{x}_i - \alpha\sum_{i=1}^{\infty}(P^{(t-1)})^i\boldsymbol{x}_i = \alpha\sum_{j=0}^{\infty}(P^{(t)})^i\boldsymbol{x}_i - \alpha\sum_{i=0}^{\infty}(P^{(t-1)})^i\boldsymbol{x}_i
\end{aligned}$$

Similarly, the second term can be formed as:

$$\begin{aligned}
&\sum_{j=0}^{\infty}(P^{(t)})^j(P^{(t)}-P^{(t-1)})\alpha\sum_{i=0}^{\infty}(P^{(t-1)})^i\boldsymbol{r}_i^{(t-1)} \\
&= \alpha\sum_{j=0}^{\infty}(P^{(t)})^i\boldsymbol{r}_i^{(t-1)} - \alpha\sum_{i=0}^{\infty}(P^{(t-1)})^i\boldsymbol{r}_i^{(t-1)}
\end{aligned}$$

Then Equ. 9 is further expressed as:

$$\hat{h}^{(t)} = \alpha \sum_{j=0}^{\infty} (P^{(t)})^i \boldsymbol{x}_i - \alpha \sum_{i=0}^{\infty} (P^{(t-1)})^i \boldsymbol{x}_i - \alpha \sum_{j=0}^{\infty} (P^{(t)})^i \boldsymbol{r}_i^{(t-1)} + \alpha \sum_{i=0}^{\infty} (P^{(t-1)})^i \boldsymbol{r}_i^{(t-1)} + \alpha \sum_{i=0}^{\infty} (P^{(t-1)})^i \boldsymbol{x}_i - \alpha \sum_{i=0}^{\infty} (P^{(t-1)})^i \boldsymbol{r}_i^{(t-1)} = \alpha \sum_{j=0}^{\infty} (P^{(t)})^i \boldsymbol{x}_i - \alpha \sum_{j=0}^{\infty} (P^{(t)})^i \boldsymbol{r}_i^{(t-1)}$$

Since the $\boldsymbol{r}_i^{(t-1)}(u) < \epsilon$ for each $u \in \mathcal{V}^{(t-1)}$, the proof is finished. □

A.7 Proof of Lemma 3

Proof. We first discuss the simple situation where only one edge (u, v) is added into graph $\mathcal{G}^{(t)}$ and transfer it into $\mathcal{G}^{(t+1)}$. Then we can naturally derive the propagation complexity of a single dimension as $\|(1-\alpha)\left(\boldsymbol{P}^{(t+1)} - \boldsymbol{P}^{(t)}\right)\boldsymbol{h}^{(t)}\|_1/\alpha\epsilon$, where $\boldsymbol{h}^{(t)}$ is one dimension of $\boldsymbol{H}^{(t)}$. As demonstrated in Alg. 3, given an added edge (u, v) the residue of $\|(1-\alpha)\left(\boldsymbol{P}^{(t+1)} - \boldsymbol{P}^{(t)}\right)\boldsymbol{h}^{(t)}\|_1$ can be divided as:

$$\begin{aligned}
&\|(1-\alpha)\left(\boldsymbol{P}^{(t+1)} - \boldsymbol{P}^{(t)}\right)\boldsymbol{h}^{(t)}\|_1 = \left|(1-\alpha)\boldsymbol{h}^{(t)}(u)/|\mathcal{N}_{out}^{(t+1)}(u)|\right| \\
&+ \left|\sum_{w \in \mathcal{N}_{out}^{(t+1)}(u)} (1-\alpha)\boldsymbol{h}^{(t)}(u)\left(1/|\mathcal{N}_{out}^{(t+1)}(u)| - 1/|\mathcal{N}_{out}^{(t)}(u)|\right)\right| \\
&\leq (1-\alpha)|\boldsymbol{h}^{(t)}(u)| + \left||\mathcal{N}_{out}^{(t+1)}(u)| \cdot (1-\alpha)\boldsymbol{h}^{(t)}(u) \cdot 1/|\mathcal{N}_{out}^{(t+1)}(u)|\right| \\
&\overset{(1)}{\leq} 2(1-\alpha)|\boldsymbol{\phi}^{(t)}(u)| + 2(1-\alpha)\epsilon,
\end{aligned}$$

where $\boldsymbol{\phi}^{(t)} = \alpha\left(I - (1-\alpha)\boldsymbol{P}^{(t)}\right)^{-1}\boldsymbol{x}^{(t)}$ and (1) holds since $\boldsymbol{h}^{(t)}(u)$ is the underestimation of $\boldsymbol{\phi}^{(t)}(u)$. Under pattern (i), edges arrive in the system randomly so that each node has the same probability of being the start point of the changed edge. Therefore we can obtain the expected time for each update as $E[\frac{2(1-\alpha)|\boldsymbol{\phi}^{(t)}(u)|+2(1-\alpha)\epsilon}{\alpha\epsilon}] = \frac{2(1-\alpha)||\boldsymbol{x}^{(t)}||_1}{\epsilon n^{(t)}} + \frac{2(1-\alpha)}{\alpha}$. Given there exist p edge updates in a batch at time t_k, we add the superscript of $\boldsymbol{x}^{(t_k)}$ and formulate the complexity for propagation as $O(p\sum_{i=1}^{F}\frac{||\boldsymbol{x}_i^{(t_k)}||_1}{\epsilon n^{(t_k)}})$. Under pattern (ii), we have $\frac{2(1-\alpha)|\boldsymbol{\phi}^{(t)}(u)|}{\alpha\epsilon} \leq \frac{2(1-\alpha)\max_{u\in\mathcal{V}}|\boldsymbol{x}^{(t)}(u)|}{\epsilon}$, which indicates the complexity of this case is $O(p\sum_{i=1}^{F}\frac{\max_{u\in\mathcal{V}}|\boldsymbol{x}_i^{(t_k)}(u)|}{\epsilon})$ □

A.8 Proof of Lemma 4

Proof. Based on Alg. 1, for an edge update (u, v), the increased mass of σ is at most $\frac{1-\alpha}{\alpha}\left\|\left(\boldsymbol{P}^{(t)} - \boldsymbol{P}^{(t-1)}\right)\cdot\boldsymbol{x}_{max}\right\|_1 + 2n\epsilon$. Following the derivation of Lemma 3, we can naturally obtain the upper bound of this term as $2(1-\alpha)|\boldsymbol{x}_{max}(u)| + 2n\epsilon$. Therefore, the largest number of edges contained before $\sigma > \lambda$ is at least $\lambda/(2(1-\alpha)|\boldsymbol{x}_{max}(u)| + 2n\epsilon)$. Then given p update events, the number of embeddings sampled L can be bounded as $L \leq (2(1-\alpha)|\boldsymbol{x}_{max}(u)| + 2n\epsilon)p/\lambda$. Finally, the time complexity of updating the node state in SSM is $O(n^{(t)}LF^2)$

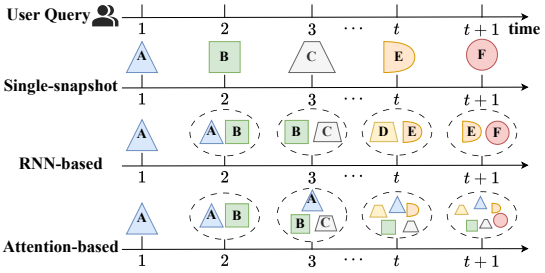


Figure 5: The comparison between different TGNN paradigms, where each color denotes a different status of the graph.

[56, 66]. Therefore, we can naturally obtain the complexity of updating node states as $O(\frac{||\boldsymbol{x}_{max}||_1+\epsilon\left(n^{(t)}\right)^2}{\lambda}pF^2)$ under pattern (i) and $O(\frac{\max_{u\in\mathcal{V}^{(t)}}|\boldsymbol{x}_{max}(u)|n^{(t)}+\epsilon\left(n^{(t)}\right)^2}{\lambda}pF^2)$ under pattern (ii). □

A.9 Related Work

A.9.1 Single-snapshot Methods. The single-snapshot GNN methods directly utilize the node embedding as the node states, which will be rapidly updated based on the new interactions. Specifically, single-snapshot GNN methods aim to transmit the updates into node embeddings $\boldsymbol{H}^{(t+1)}$ from $\boldsymbol{H}^{(t)}$ with the minimal time cost following:

$$\boldsymbol{H}^{(t+1)} = \text{MSG}\left(\mathcal{G}^{(t)}, \boldsymbol{H}^{(t)}, e_{t+1}\right). \tag{10}$$

To facilitate the incremental updating based on these pre-computed node embeddings, Instant [32], DynAnom [33] and IDOL [42] explore the invariant rules of graph propagation and conduct an invariant-based algorithm of Personalized PageRank (PPR) to refresh the node embedding locally. The dominant update complexity of these methods can be bounded as $O(p\sum_{i=1}^{F}\frac{||\boldsymbol{x}_i^{(t)}||_1}{\epsilon n^{(t)}})$ given p updates and ϵ approximation error [32]. However, only focusing on the current snapshot will miss significant interactions of past time steps, leading to sub-optimal prediction results.

A.9.2 RNN-based Methods. RNN-based methods are generally based on the classical RNN [52], GRU [53], LSTM [54] algorithm, etc., which simply use both the current input and the previous hidden state to iteratively capture temporal dependencies. For example, the GRU algorithm updates the node states at time $t+1$ as:

$$\begin{aligned}
\boldsymbol{Z}^{(t+1)} &= \text{sigmoid}\left(\boldsymbol{W}_Z\boldsymbol{H}^{(t+1)} + \boldsymbol{U}_Z\boldsymbol{M}^{(t)} + \boldsymbol{B}_Z\right) \\
\boldsymbol{R}_t &= \text{sigmoid}\left(\boldsymbol{W}_R\boldsymbol{H}^{(t+1)} + \boldsymbol{U}_R\boldsymbol{M}^{(t)} + \boldsymbol{B}_R\right) \\
\widetilde{\boldsymbol{H}}^{(t+1)} &= \tanh\left(\boldsymbol{W}_H\boldsymbol{H}^{(t+1)} + \boldsymbol{U}_H\left(\boldsymbol{R}_t \circ \boldsymbol{M}^{(t)}\right) + \boldsymbol{B}_H\right) \\
\boldsymbol{M}^{(t+1)} &= \left(1 - \boldsymbol{Z}^{(t+1)}\right) \circ \boldsymbol{M}^{(t)} + \boldsymbol{Z}^{(t+1)} \circ \widetilde{\boldsymbol{H}}^{(t+1)},
\end{aligned}$$

where $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{B}$ denote the trainable parameters of the linear layer and $\boldsymbol{H}^{(t+1)}$ can be updated using the MSG$(\cdot)$. Specifically, TGCN [44] and EvolveGCN [34] and incorporate the Graph Convolutional Network (GCN)[55] as the MSG$(\cdot)$ function to regenerate the node embeddings while coupling with an RNN-based module to learn temporal node representations. Similarly, MPNN [35] transforms the node embeddings at different time steps into an RNN module and then captures the long-range dependency in the final hidden

state. This category of method generally requires $O(Km^{(t)}F)$ and $O(pn^{(t)}F^2)$ to update the node embeddings and states [5], respectively. Due to their iterative structure, RNN-based methods can efficiently update node states. However, the simplistic recurrence mechanism often leads to difficulties in retaining historical information, especially as the graph size and temporal scope expand [56, 57].

A.9.3 Attention-based Methods. To address the forgetting problem of RNNs, attention-based methods rely on the attention mechanism and abstain from using recurrence form, which encodes the position of sequences and enables the efficient information flow from past to current representations. We take the representative work APAN [84] as an example to demonstrate the core mechanism of this category. Considering matrices $\boldsymbol{Q} \in \mathbb{R}^{n\times F}$ denoted as "query", $\boldsymbol{K} \in \mathbb{R}^{n\times F}$ denoted as "keys", and $\boldsymbol{V}^{n\times F}$ denoted as "values", the classical attention algorithms perform the following computation to obtain the optimized embeddings:

$$\boldsymbol{M}^{(t+1)} = \text{Attn}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{F}}\right)\boldsymbol{V},$$
$$\boldsymbol{Q} = \boldsymbol{H}^{(t+1)}\boldsymbol{W}_q, \boldsymbol{K} = \boldsymbol{M}^{(t)}\boldsymbol{W}_k, \boldsymbol{V} = \boldsymbol{M}^{(t)}\boldsymbol{W}_v,$$

where $\boldsymbol{W}_q, \boldsymbol{W}_k, \boldsymbol{W}_\in \mathbb{R}^{F\times F'}$ are the network parameters. The dot-product term $\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{F}}\right)$ takes the role of weighting the interactions between entity "query-key" pairs. A higher value within this term increases the contribution of $\boldsymbol{V}$ to the embedding space. Thus, attention-based methods create the expressive attention score to capture the relationship between the current embedding and the state of the last time step. Following this intuition, DySat [37] employs the generalized GAT module [58] to integrate the embeddings of a single node from different time steps to generate its refreshed one. ASTGCN [38] and DNNTSP [46] further incorporate the attention mechanism to capture the spatial and temporal dependency for enhanced representation quality. For each time step, these methods generally require $O(T\left(n^{(t)}\right)^2 F)$ time complexity to calculate the final output given T time step. While attention mechanisms can retain the most relevant parts of the sequences to avoid the forgetting issue, they can become computationally expensive with frequent updates [59].

A.9.4 Other TGNN methods. (i) *SNN-based methods.* Another typical mechanism of this category is based on the biological Spiking Neural Networks (SNNs), which simulate the brain behaviors and maintain the membrane potential given the data sequences. SpikeNet [1] retrieves the node embeddings from multiple time steps and finally generates the prediction results by the spike firing process. Dy-SIGN [85] incorporates SNNs mechanism into dynamic graphs to mitigate the information loss and memory consumption problem. Nevertheless, the demand for multiple simulation steps to generate reliable embeddings can significantly degrade the efficiency of these TGNN methods. (ii) *SSM-based methods.* There are also some works which employs the SSM mechanism on temporal graphs. For example, STG-Mamba [69] formulates the feature stream of each node as the long-term context, which improves the embedding quality for feature-varying graphs. Graph-SSM [56] addresses the unobserved graph mutations between consecutive snapshots, and achieves an effective discretization with long-term information. However, these methods only focus on the discrete-time dynamic graph and fail to model the continuous topology changes as the graph evolves. Furthermore, directly adapting these methods will incur significant computational overhead as the graph evolves, creating a gap between current algorithms and continuous prediction in practical dynamic scenarios.

A.9.5 State Space Models. In recent years, structured State Space Models (S4) [68, 86] have emerged as promising frameworks for modeling long-distance sequences, offering the advantage of only a linear increase in computational cost. Given the input series $x(t) \in \mathbb{R}$, structured State Space Models (S4) [68, 86] formulates the state variable series $h(t) \in \mathbb{R}$ and the output series $y(t) \in \mathbb{R}$ using the following equations:

$$\begin{cases} h'(t) &= \mathcal{A}h(t) + \mathcal{B}x(t) \\ y(t) &= Ch(t), \end{cases} \tag{11}$$

$$\begin{cases} h_t &= \bar{\mathcal{A}}h_{t-1} + \bar{\mathcal{B}}x_t \\ y_t &= Ch_t, \end{cases} \tag{12}$$

where $(\mathcal{A}, \mathcal{B}, C, \bar{\mathcal{A}}, \bar{\mathcal{B}})$ are trainable parameters. Here Equ. 11 and 12 are the continuous and discretization process, respectively. Compared with the RNN-based or attention-based mechanism, S4's strength lies in its adherence to a linear mechanism, which guarantees enhanced stability control [56]. This facilitates effective long-term modeling of sequences through meticulous initialization of state space layer parameters [87, 88]. As a result, we employ the SSMs as our temporal-processing unit to meticulously balance the accuracy and efficiency of our CODEN framework.

A.9.6 Forward Push. The prevailing paradigm for existing PPR-based propagation is derived from the concept of *Forward Push*, which calculates the approximated solution $\boldsymbol{z}_i = \sum_{l=0}^{\infty} \alpha(1-\alpha)^l \left(\boldsymbol{A}^{(t)\top}\boldsymbol{D}^{(t)-1}\right)^l \boldsymbol{x}_i^{(}$ $(1 \le i \le F)$ under a given error bound ϵ, where α is the decay factor of random walk. Specifically, *Forward Push* (depicted in Alg. 2) assigns the attribute vector $\boldsymbol{x}_i^{(t)}$ to the *residue vector* $\boldsymbol{r}^{(t)}$ (e.g., $\boldsymbol{r}^{(t)} = \boldsymbol{x}_i^{(t)}$), which represents the unpropagated mass of attribute vector $\boldsymbol{x}_i^{(t)}$. [6] We iteratively conduct the following two steps: (1) For each node $s \in \mathcal{V}^{(t)}$ such that $\boldsymbol{r}^{(t)}(s) > \epsilon$, $(1-\alpha)$ fraction of $\boldsymbol{r}^{(t)}(s)$ will be propagated into the out-neighbors $t \in \mathcal{N}_{out}(s)$ averagely. (2) $(1-\alpha)$ fraction of $\boldsymbol{r}^{(t)}(s)$ will be transferred into the reserve vector $\boldsymbol{h}^{(t)}$ and then $\boldsymbol{r}^{(t)}(s)$ is set as 0. This iteration will be terminated until $\boldsymbol{r}^{(t)}(s) \le \epsilon$ for all nodes $s \in \mathcal{V}^{(t)}$ and we can deploy $\boldsymbol{h}^{(t)}$ as the approximated node embedding which satisfies $|\boldsymbol{h}^{(t)}(s) - \boldsymbol{z}^{(t)}(s)| \le \epsilon$ for each node $s \in \mathcal{V}^{(t)}$.

A.10 Introduction of baseline methods

A.10.1 Single-snapshot methods. **Instant [32], DynAnom [33], IDOL [42].** These three methods inherit the updating skeleton of PPR [89] to incrementally update the node embeddings. Following the analysis of [32], the dominant complexity of updating embeddings can be formulated as $O\left(\frac{||\boldsymbol{x}^{(t)}||_1}{\epsilon n}\right)$ for a feature vector $\boldsymbol{x}^{(t)}$ assuming α as the constant. Hence we summarize the complexity of this catogory as $O\left(p\sum_{i=1}^{F}||\boldsymbol{x}_i^{(t)}||_1\right)$ for F dimensions given $\epsilon = \frac{1}{n}$.

[5] For a clear presentation, we assume the dimension of node state $F' = F$

[6] In the following sections, we will omit the subscript i for simplicity.

Algorithm 2: Forward Push

Input : Graph $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$, reserve vector $\boldsymbol{h}^{(t)}$, residue vector $\boldsymbol{r}^{(t)}$.
Output: Reserve vector $\boldsymbol{h}^{(t)}$

$\boldsymbol{h}^{(t)} = \boldsymbol{h}^{(t-1)}$;
while *exists* $s \in \mathcal{V}$ *such that* $\boldsymbol{r}^{(t)}(s) > \epsilon$ **do**
  foreach $v \in \mathcal{N}_{out}^{(t)}(s)$ **do**
    $\boldsymbol{r}^{(t)}(v) += (1-\alpha) \cdot \frac{\boldsymbol{r}^{(t)}(s)}{|\mathcal{N}_{out}^{(t)}(s)|}$;
  $\boldsymbol{h}^{(t)}(s) += \alpha \cdot \boldsymbol{r}^{(t)}(s)$; $\boldsymbol{r}^{(t)}(s) = 0$;

A.10.2 RNN-based methods. **TGN [43].** TGN proposes a general framework for learning the representation in CTDG, which models the past interactions between nodes using a compressed node state vector. Based on existing techniques, TGN provides flexible modules to compute the embeddings and update node states. For example, TGN utilizes the attention mechanism and GCN [55] to compute the node embeddings and LSTM or GRU module to update the node state.

TGCN [44], EvolveGCN [34], MPNN [35], ROLAND [45]. These four methods employ a unified pipeline to update the node embeddings and states. Specifically, these methods conduct the graph propagation based on GCN [55] and update the state with GRU or LSTM units. Since each update will require to compute the embeddings from scratch, we summarize their complexity as $O(Km^{(t)}F)$ and $O(pn^{(t)}F^2)$ given p updates.

A.10.3 Attention-based methods. **DySat [37], ASTGCN [38].** DySat and ASTGCN utilize structural and temporal attention mechanism for dynamic graph representation learning. Both of them employ graph propagation by GCN. As a result, the graph propagation and state update require $O(n^2F)$ time complexity for each update.

TGAT [24]. TGAT adopts the GraphSage [60] for the embedding computation. Different from DySat, TGAT aims to formulate the interactions between the time encoding and node features. However, due to the polynomial property of the attention mechanism, TGAT still needs to consume $O(n^2F)$ time for state update.

DNNTSP [46], SEIGN[47], DyGFormer [48]. These three methods follow a similar intuition to our alternative model CODEN-A, which intends to calculate the interactions between different time steps for each node. Given the time step t, this framework needs to consume $O(tn^2F)$ time for the state update.

A.11 Accuracy-guaranteed Embedding Update

As we mentioned, our objective is to efficiently propagate frequent updates into the node embeddings from the preceding time step. The rationale behind this objective is that not all node embeddings can be significantly influenced by the updates, since an edge update causes only a minimal impact on distant nodes. However, current state-of-the-art algorithms [1, 40, 43] intricately intertwine graph convolution with node states utilizing trainable parameters in the message function MSG$(\cdot)$ of Equ. 1, requiring either repetitive convolution operations or a complete recalculation of states upon updates. To address these issues, we propose to incrementally integrate update events into the node embeddings $\boldsymbol{h}^{(t)}$ in the subsequent sections. This step will then trigger an update to the node states $\boldsymbol{M}^{(t)}(u)$ to effectively document the ongoing evolutionary process.

Algorithm 3: Embedding Update

Input : Graph $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$, update events $\Gamma = \{(u_1, v_1), (u_2, v_2), ..., (u_p, v_p)\}$, reserve vector $\boldsymbol{h}^{(t)}$, old adjacent matrix $\boldsymbol{P}^{(t)}$, updated adjacent matrix $\boldsymbol{P}^{(t+p)}$.
Output: Reserve vector $\boldsymbol{h}^{(t+p)}$

$\boldsymbol{r}^{(t+p)} = \boldsymbol{0}$;
$\mathcal{V}_{changed} \leftarrow \{u|$ the nodes whose out degree has changed$\}$;
foreach $u \in \mathcal{V}_{changed}$ **do**
  $\mathcal{N}_{add}(u) \leftarrow \{v|$ the added neighbors of $u\}$;
  $\mathcal{N}_{del}(u) \leftarrow \{v|$ the deleted neighbors of $u\}$;
  foreach $v \in \mathcal{N}_{add}(u)$ **do**
    $\boldsymbol{r}^{(t+p)}(v) += (1-\alpha)\boldsymbol{h}^{(t)}(u)/|\mathcal{N}_{out}^{(t+p)}(u)|$;
  foreach $v \in \mathcal{N}_{del}(u)$ **do**
    $\boldsymbol{r}^{(t+p)}(v) -= (1-\alpha)\boldsymbol{h}^{(t)}(u)/|\mathcal{N}_{out}^{(t)}(u)|$;
  foreach $w \in \mathcal{N}_{out}^{(t+p)}(u) \setminus (\mathcal{N}_{add}(u) \cup \mathcal{N}_{del}(u))$ **do**
    $\boldsymbol{r}^{(t+p)}(w) += (1-\alpha)\boldsymbol{h}^{(t)}(u)\left(\frac{1}{|\mathcal{N}_{out}^{(t+p)}(u)|} - \frac{1}{|\mathcal{N}_{out}^{(t)}(u)|}\right)$;
Forward Push $(\mathcal{G}^{(t+p)}, \boldsymbol{h}^{(t)}, \boldsymbol{r}^{(t+p)})$;

Compensated Propagation. In this section, we discuss how to efficiently update the node embedding $\boldsymbol{H}^{(t)}$ and let $||\boldsymbol{H}^{(t)} - \boldsymbol{Z}^{(t)}||_1 \leq n^{(t)}\epsilon$ holds for each time step t as demonstrated in 1. Despite of the recent advancements to incrementally update the PPR-based embeddings [32, 33], it is pointed out that this invariant-based adjustment can not provide the accuracy bound after updating [65, 90]. To address this challenge, we draw on insights from recent works on Random Walk with Restart (RWR) [65] to compensate for embedding distance across different time steps.

Without loss of generality, we start with an example that $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} \in \boldsymbol{X}^{(t)}$ and a new edge $e_{t+1} = (u, v)$ is added at time $t+1$, creating an unbounded value difference between the old embedding $\boldsymbol{h}^{(t)}$ and the approximation target $\boldsymbol{z}^{(t+1)}$. Note that $\boldsymbol{h}^{(t)}$ is the approximation of $\boldsymbol{z}^{(t)}$, and we have:

$$\begin{aligned}
\boldsymbol{z}^{(t+1)} - \boldsymbol{z}^{(t)} &= \alpha\left(\left(I-(1-\alpha)\boldsymbol{P}^{(t+1)}\right)^{-1} - \left(I-(1-\alpha)\boldsymbol{P}^{(t)}\right)^{-1}\right)\boldsymbol{x}^{(t+1)} \\
&\overset{(1)}{=} \alpha(1-\alpha)\left(I-(1-\alpha)\boldsymbol{P}^{(t+1)}\right)^{-1}\Delta\boldsymbol{P}\left(I-(1-\alpha)\boldsymbol{P}^{(t)}\right)^{-1}\boldsymbol{x}^{(t+1)} \\
&= \underbrace{\alpha\left(I-(1-\alpha)\boldsymbol{P}^{(t+1)}\right)^{-1}}_{\textit{propagation process}} \cdot \underbrace{(1-\alpha)\Delta\boldsymbol{P}\boldsymbol{z}^{(t)}}_{\textit{compensated vector}}
\end{aligned} \tag{13}$$

where (1) holds since $\boldsymbol{U}^{-1} - \boldsymbol{V}^{-1} = \boldsymbol{U}^{-1}(\boldsymbol{V}-\boldsymbol{U})\boldsymbol{V}^{-1}$ and $\Delta\boldsymbol{P} = \left(\boldsymbol{P}^{(t+1)} - \boldsymbol{P}^{(t)}\right)$. Based on Equ. 13, we have a key observation: *the difference between node embedding* $\boldsymbol{z}^{(t)}$ *and* $\boldsymbol{z}^{(t+1)}$ *can be compensated by propagating the new feature vector* $(1-\alpha)\left(\boldsymbol{P}^{(t+1)} - \boldsymbol{P}^{(t)}\right)\boldsymbol{z}^{(t)}$ *along the updated graph* $\mathcal{G}^{(t+1)}$. Since $\boldsymbol{h}^{(t)}$ is the approximation of $\boldsymbol{z}^{(t)}$ with the allowed error, we hence propose to implement the embedding update based on the *compensated vector* $(1-\alpha)\Delta\boldsymbol{P}\boldsymbol{h}^{(t)}$.

Scalable Batch Update. By adhering to the principle above, we extend the incremental update of node embedding into the batch setting, which is depicted in Alg. 3. Consider the graph $\mathcal{G}^{(t)}$ at time t and the batch update events in $\Gamma = \{(u_1, v_1), (u_2, v_2), ..., (u_p, v_p)\}$

containing p edge updates, where (u_i, v_i) $(1 \leq i \leq p)$ will be viewed as deletion if it exists in $\mathcal{G}^{(t+i-1)}$ otherwise as an addition. First, we obtain the set $\mathcal{V}_{changed}$ denoting the nodes whose out-degree has changed. Given a node $u \in \mathcal{V}_{changed}$ and for each node $v \in \mathcal{N}_{out}^{(t)}(u) \cup \mathcal{N}_{out}^{(t+p)}(u)$, the corresponding entry of $\Delta \boldsymbol{P} = \boldsymbol{P}^{(t+p)} - \boldsymbol{P}^{(t)}$ can be expressed as:

$$\Delta \boldsymbol{P}[v,u] = \begin{cases} \frac{1}{|\mathcal{N}_{out}^{(t+p)}(u)|}, & v \text{ is the added neighbor of } u, \\ -\frac{1}{|\mathcal{N}_{out}^{(t)}(u)|}, & v \text{ is the deteted neighbor of } u, \\ \frac{1}{|\mathcal{N}_{out}^{(t+p)}(u)|} - \frac{1}{|\mathcal{N}_{out}^{(t)}(u)|}, & \text{otherwise.} \end{cases}$$

Then, we multiply $\Delta \boldsymbol{P}[v,u]$ with $(1-\alpha)\boldsymbol{h}^{(t)}(u)$ and finish the computation of the compensated vector $(1-\alpha)\left(\boldsymbol{P}^{(t+p)} - \boldsymbol{P}^{(t)}\right)\boldsymbol{h}^{(t)}$ (lines 6-11). Moreover, we assign this new feature vector as the unpropagated residue vector $\boldsymbol{r}^{(t+p)}$. Note that $\boldsymbol{r}^{(t+p)}$ may exceed the permissible error for certain nodes, such as when $\boldsymbol{r}^{(t+p)}(u) > \epsilon$ $(u \in \mathcal{V}^{(t+p)})$. Consequently, we trigger the *Forward Push* mechanism [79] again to propagate the residue vector $\boldsymbol{r}^{(t+p)}$ (line 12), which will subsequently influence the embedding values of other nodes and ensure the desired accuracy. Additionally, our algorithm can also be compatible with the attribute changes, which is detailed in the Appendix [41].

Theoretical Accuracy Guarantee. Alg. 3 returns the updated embeddings by propagating the compensated vector along the updated graph, where the compensated vector is derived using $\boldsymbol{z}^{(t)}$, as specified in Equ. 13. Although we process this vector with the approximated vector $\boldsymbol{h}^{(t)}$, we can still establish an error bound for the output vector $\boldsymbol{h}^{(t+p)}$, which is the result of propagation from scratch at time $t+p$. We formally state this accuracy guarantee in the following lemma:

Lemma 5. *Given the normalized adjacent matrix $\boldsymbol{P}^{(t)}$ at time t and the update event set $\Gamma = \{(u_1, v_1), (u_2, v_2), ..., (u_p, v_p)\}$, Alg 3 can output the approximated embedding vector $\boldsymbol{h}^{(t+p)}$ which satisfies:*

$$||\boldsymbol{H}^{(t+p)} - \boldsymbol{Z}^{(t+p)}||_1 \leq n^{(t)}\epsilon$$

where $\boldsymbol{Z}^{(t+p)} = \sum_{l=0}^{\infty} \alpha(1-\alpha)^l \left(\boldsymbol{P}^{(t+p)}\right)^l \boldsymbol{X}^{(t+p)}$.

Comparison with the invariant-based scheme. Compared with the invariant-based scheme in existing related works [32, 33, 42], we claim that our method stands out as the first to offer guaranteed accuracy of updated embeddings in dynamic settings. The rationale for this difference is that our method offers a more stable adjustment by leveraging a compensate vector, making it more robust in scenarios with significant node degree changes. A running example underpinning this insight is further provided in the Appendix [41]. In summary, our accuracy-guaranteed embedding update forms the novel theoretical foundation of Coden (e.g., Lemma 1 and Proposition 1), thereby conferring substantial benefits in producing high-quality representations.

A.12 Extension to attribute changes.

Note that we have made an assumption that $\boldsymbol{x}^{(t+p)} = \boldsymbol{x}^{(t)}$ such that we can focus on the topological change of the graph. Actually, this assumption can be easily relaxed since extending our approach

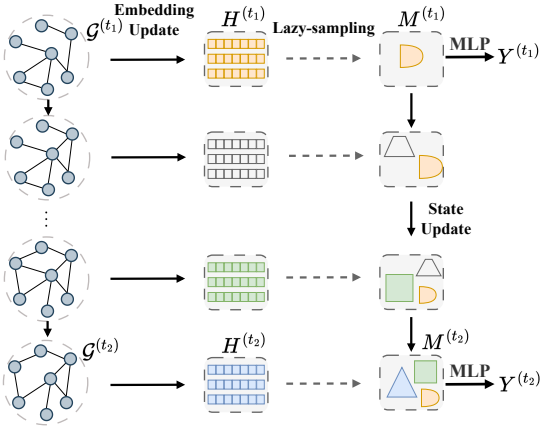


Figure 6: An illustration of the Coden framework. $\boldsymbol{M}^{(t)}$, $\boldsymbol{H}^{(t)}$ and $\boldsymbol{Y}^{(t)}$ denote the node state matrix, embedding matrix and the prediction results at time t, respectively.

to dynamic-attribute graphs is much more straightforward. Specifically, given $\Delta \boldsymbol{x} = \boldsymbol{x}^{(t+p)} - \boldsymbol{x}^{(t)}$ and $\boldsymbol{P}^{(t+p)} = \boldsymbol{P}^{(t)}$, we can naturally derive the difference between $\boldsymbol{z}^{(t)}$ and $\boldsymbol{z}^{(t+p)}$ as:

$$\boldsymbol{z}^{(t+p)} - \boldsymbol{z}^{(t)} = \alpha \left(I - (1-\alpha)\boldsymbol{P}^{(t+1)}\right)^{-1} \Delta \boldsymbol{x}. \tag{14}$$

Therefore, we can view $\Delta \boldsymbol{x}$ as the compensated vector and easily obtain the updated embedding by directly invoking *Forward Push* $(\mathcal{G}^{(t+p)}, \boldsymbol{h}^{(t)}, \Delta \boldsymbol{x})$. [7]

A.13 Discussion of the Invariant-based Scheme

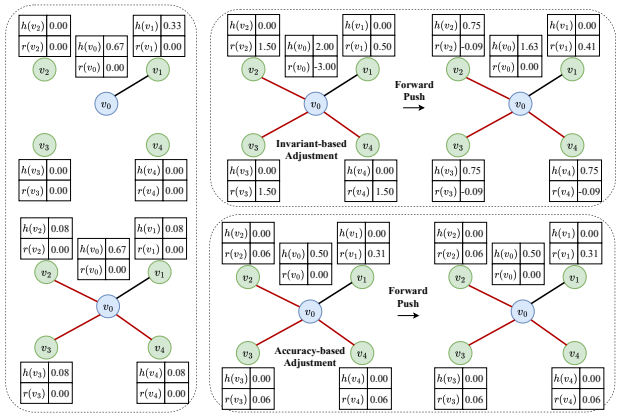
Considering p edge updates, existing related works [32, 33, 42] adhere the invariant relationship in PPR and perform the following rules to locally update the node embedding: $\boldsymbol{r}^{(t+p)}(u) \leftarrow \boldsymbol{r}^{(t)}(u) - \frac{\Delta_p(u)\boldsymbol{h}^{(t)}(u)}{\alpha|\mathcal{N}_{out}^{(t)}(u)|}, \boldsymbol{r}^{(t+p)}(v) \leftarrow \boldsymbol{r}^{(t)}(v) + \frac{(1-\alpha)\Delta_p(u)\boldsymbol{h}^{(t)}(u)}{\alpha|\mathcal{N}_{out}^{(t)}(u)|}, \boldsymbol{h}^{(t+p)}(u) \leftarrow \boldsymbol{h}^{(t)}(u) \cdot \frac{|\mathcal{N}_{out}^{(t+p)}(u)|}{|\mathcal{N}_{out}^{(t)}(u)|}$, where $\Delta_p(u) = |\mathcal{N}_{out}^{(t+p)}(u)| - |\mathcal{N}_{out}^{(t)}(u)|$. The *Forward Push* process is then repeated to ensure that local adjustments propagate to distant nodes. Although these methods follow a similar pipeline to Coden, we emphasize that our initialization process prior to *Forward Push* is fundamentally different.

To illustrate this distinction, we present a toy example in Fig. 7 where 3 edges are added between node v_0 and $v_2 \sim v_4$. Based on this result, we see that the invariant-based scheme can not guarantee the accuracy of node embeddings since $|\boldsymbol{h}(v) - \boldsymbol{z}(v)| = |0.75 - 0.08| > 0.5 = \epsilon$ for $v = v_2 \sim v_4$. In contrast, our accuracy-based can still maintain the embedding accuracy after the *Forward Push* algorithm, which confers substantial benefits in producing high-quality representations for later node state formulation.

A.14 Additional Experimental Results

A.14.1 Experimental Setting . Since most of the compared RNN-based and attention-based methods are designed for small-scale graphs, we adopt the neighboring sampling techniques [60] for these methods when applied on the *Reddit* and *Patent* datasets to avoid the out-of-memory problem. We summarize the experimental

[7] In the remaining sections, we mainly focus on the topology change and assume $\boldsymbol{X}^{(t)}$ remains unchanged in the following sections for a clear presentation.



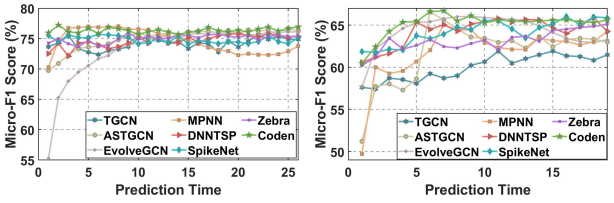
(a) *Propagation from the scratch* (b) *Result of variant-based and accuracy-based adjustment*

Figure 7: A toy example to compare the results of 4 kinds of push operations when we set $\alpha = 0.5$ and $\epsilon = 0.5$. (a) shows the initial push result before the update. (b) shows the ground truth result of the push from scratch. (c) shows the result of the variant-based push, where the error of the embeddings of node $v_2 \sim v_4$ have exceeded the bound ϵ. (d) shows the result of the accuracy-based push, where the accuracy of node embeddings is guaranteed.

Table 7: Parameter settings. Here "lr" means the learning rate, "K" means the number of convolution layers, "hidden" means the hidden size of the network, and "batch number" means the number of neighbor sampling for baselines.

Datasets	lr	K	hidden	batch number
DBLP	1e-3	4	1024	12
Tmall	1e-3	4	1024	12
Reddit	1e-3	4	512	12
Patent	1e-3	4	512	12
Papers100M	1e-3	2	512	12

settings for all baselines in Tab. 7, where the common parameters, such as learning rate and hidden size, are maintained consistently for CODEN.

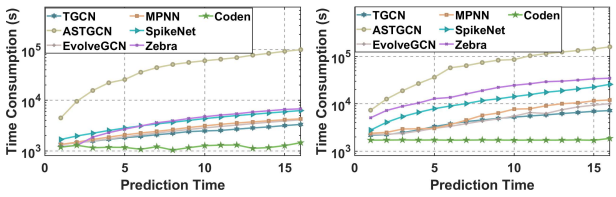


(a) *DBLP* (b) *Tmall*

Figure 8: Micro-F1 scores for each prediction time.

A.14.2 Micro-F1 scores and training time on DBLP and Patent.

A.14.3 Information compression under flexible arrival patterns. In the experiments above, we added edges to the initial graph following an edge-inclined arrival pattern. However, using a more flexible arrival pattern would allow for a more thorough examination of the models' memory capabilities, particularly in assessing whether



(a) *Reddit* (b) *Patent*

Figure 9: Training time consumption for each prediction time.

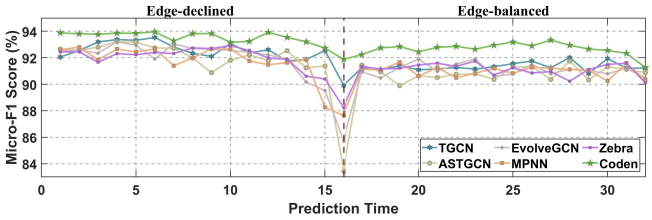
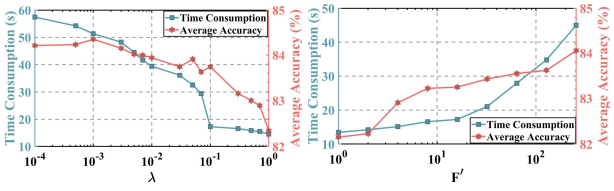


Figure 10: Micro-F1 scores under the Edge-declined and Edge-balanced patterns on *Reddit* dataset.

they experience significant performance degradation. Therefore, we employ more edge arrival patterns on each model and report the corresponding accuracy on *Reddit* dataset, as depicted in Fig. 10. In the edge-declined pattern, we reverse the order of the edge-inclined setting and remove the batch of edges sequentially from the final snapshot. Then, we randomly add and remove one batch of edges simultaneously to the initial graph, which is denoted as the edge-balanced pattern. In the edge-declined pattern, we observe that as the edges are missing gradually, all methods suffer from a performance decrease. However, CODEN is only slightly affected by the removal of edge batches, which particularly achieves a non-trivial improvement on the initial graph compared with the edge-inclined pattern at time step 16 (e.g., 91.86% vs 87.86%). Moreover, CODEN demonstrates a significant improvement in this pattern, achieving an average prediction accuracy of 93.48%, indicating its ability to effectively compress historical information for future predictions.



(a) Patent

Figure 11: The average training time and the average accuracy when setting different λ and F' on *Patent* dataset.